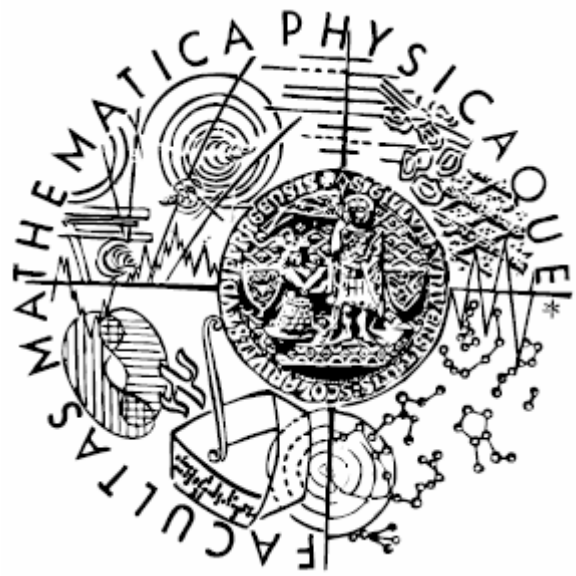
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta



Bakalárska práca

Martin Brehovský

Podpora navrhovania relačnej schémy pre databázu

Kabinet software a výuky informatiky

Vedúci bakalárskej práce:
Mgr. Jaroslav Semančík

Študijný program: Informatika
Správa počítačových systémov

2006

Na tomto mieste by som rád poďakoval vedúcemu bakalárskej práce Mgr. Jaroslavovi Semančíkovi za cenné rady a podporu v priebehu jej tvorby. Taktiež ďakujem všetkým ľuďom, ktorí mi pomáhali s týmto neľahkým projektom či už testovaním aplikácie, alebo korektúrou textu.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa 22.4.2006 Martin Brehovský

Obsah

Názov práce: Podpora navrhovania relačnej schémy pre databázu
Autor: Martin Brehovský
Katedra (ústav): Kabinet software a výuky informatiky
Vedúci bakalárskej práce: Mgr. Jaroslav Semančík
e-mail vedúceho: Jaroslav.Semancik@mff.cuni.cz

Abstrakt: Práca „Podpora navrhovania relačnej schémy pre databázu“ poukazuje na dôležitosť a problémy spojené s návrhom databázových schém v súčasnosti. Zobrazuje na konkrétnych príkladoch možné algoritmy poskytujúce ich riešenie. Účelom práce je predniesť teoreticky ucelenú prácu o možnostiach analýzy a správneho návrhu relačných schém. Tieto schémy musia spĺňať požiadavky na konzistenciu dát a ich bezproblémovú správu. Hlavnou súčasťou práce je aplikácia, ktorá implementuje algoritmy efektívne riešiace tieto problémy pri návrhu.

Kľúčové slová: dekompozícia, syntéza, uzáver, normálna forma

Title: Computer aided design of database relation schemes
Author: Martin Brehovský
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Jaroslav Semančík
Supervisor’s e-mail address: Jaroslav.Semancik@mff.cuni.cz

Abstract: Project “Computer aided design of database relation schemes” shows the importance and problems with designing nowadays Database schemas and gives ideas on possible solutions to these problems. Its main purpose is to describe a solution on optimizing Database designing that is transforming database schemas into tables. These tables should also fulfill the requirements based on the need for data consistency and database maintenance. Main part of this project is an application, which implements algorithms for solving these problems in an efficient way.

Keywords: decomposition, synthesis, closure, normal form

Kapitola 1

Úvod

1.1 Cieľ

Hlavným účelom tejto práce je vytvoriť program a zdokumentovať teóriu, ktorá pomôže užívateľovi pri navrhovaní schém databáz. Program je určený k optimalizácii ich navrhovania. Štandardná databáza je zložená z množstva tabuliek. Pri návrhu databázy sa najprv určia isté závislosti medzi jednotlivými atribútmi a z nich potom odvodíme jednotlivé tabuľky databázy. Otázkou ostáva, či je tento návrh efektívny, respektíve, či nie je možné ho vylepšiť. Problémom je, ako overiť správnosť a zefektívniť prácu s danou databázou a následne navrhnúť jej architektúru. Tento program je určený na hľadanie optimálneho riešenia.

Zmyslom a účelom programu nie je optimalizácia triviálnych databázových tabuliek, ale najmä nájdenie optimálneho návrhu pre pomerne veľké databázy, ktorých štruktúra nie je úplne jasná.

V teoretickej časti sa práca zameriava na popis základných algoritmov použitých na normalizáciu schémy a ďalších, slúžiacich k ich analýze. Načrtnutím konkrétneho problému a následným návrhom jeho riešenia má čitateľ možnosť na praktických príkladoch vidieť dôležitosť optimalizácie a nevyhnutnosť normalizácie schém.

1.2 Popis problému

V tejto časti sú na jednoduchých príkladoch popísané problémy, ktoré môžu nastať pri nesprávnom navrhnutí databázovej schémy. V takýchto prípadoch môže dochádzať k aktualizačným anomáliám, ktoré sa prejavujú použitím operácií INSERT, DELETE a UPDATE. Sú to najčastejšie javy, pri ktorých dochádza k nadbytočnosti či duplicite informácií. V tomto prípade tabuľka zaberá zbytočne veľa miesta a môže nastať problém s jej údržbou. Nežiaduca strata dát pri mazaní čiastkových informácií, či znemožnenie ukladania, ak dáta nie sú kompletné, sú problémy, súvisiace s nesprávnym návrhom a motiváciou na hľadanie iných možností na ich uloženie.

Na uvedených príkladoch sú popísané jednotlivé anomálie. Navrhnuté spôsoby ich riešenia optimalizujú návrh a slúžia na zdôraznenie potreby normalizácie schém.

1.2.1 Nadbytočnosť dát

1) Je potrebné predchádzať problému duplicitných dát. Predstavme si atribúty **F**irma, **S**idlo, **V**yrobok a **M**esto a tabuľku **FSVM** tak, že sú to všetky firmy v danom meste spolu s výrobkami a sídlom firmy. Ak sa nad tým trochu zamyslíme, dôjdeme k záveru, že táto tabuľka je zbytočne veľká, každá firma tam bude príliš veľakrát, ak vyrába veľa výrobkov. Navyše nastávajú problémy s údržbou takejto tabuľky. Ak zmeníme sídlo firmy, bude nutné ju meniť v databáze pri každom zázname o výrobku, ktorý firma vyrába.

1.2.2 Strata dát

2) Použijem už popísaný príklad s atribútmi **F**irma, **S**idlo, **V**yrobok, **M**esto a tabuľkou **FSVM.** Čo sa stane, ak sa vo firme zmení vyrábaný sortiment a na určitú dobu sa odstaví kvôli výmene strojov. Je to bežná situácia, a tak musíme na čas, kým sa nevyrába, zmeniť informácie o firme v našej databáze. Ak teda zmažeme všetky záznamy o výrobkoch produkovaných firmou, stratíme o nej aj všetky informácie.

To je nechcený jav, pretože jej sídlo ani mesto, v ktorom sa firma nachádzala, sa nezmenilo a záznam o existencii firmy má byť aj tak evidovaný.

Je však možnosť vyriešiť takúto stratu dát použitím hodnoty null namiesto výrobku, ktorá označuje, že firma tento výrobok nevyrába. Tu však nastávajú iné komplikácie. Ak sa správca rozhodne pri mazaní výrobkov iba meniť ich hodnoty na null, nastane situácia, pri ktorej sa bude v databáze nachádzať veľa nadbytočných záznamov so zhodnými informáciami, teda Firma, Sidlo, null, Mesto. Správca bude musieť tieto duplicity redukovať, čo je zjavne nepohodlné. Môže sa však rozhodnúť vykonávať mazanie iným spôsobom a to tak, že hodnotu null nastaví, iba ak je záznam o výrobku posledný. Teda ide o jediný riadok v databáze, ktorý obsahuje informácie o firme. Tento fakt by musel správca ošetrovať pri každom mazaní, a tak by sa v najhoršom prípade musela prejsť na jeho overenie celá databáza.

1.2.3 Znemožnenie ukladania dát

3) V situácii načrtnutej v predošlom príklade nastáva ešte jeden problém. Je ním znemožnenie zadávania nových informácií, ak nie sú úplné. Napríklad o novovytvorenej firme vieme jej názov, mesto a sídlo, ale informácia o výrobkoch nie je známa. Na uloženie takýchto neúplných informácií sa dá ,samozrejme, použiť ako výrobok hodnota null. Ibaže pri každom vkladaní nových výrobkov bude musieť správca testovať existenciu záznamu s hodnotou null a v prípade úspechu ju nahradiť aktuálnou informáciou. To je značne nepohodlné pre správcu a neoptimálne pre databázu.

1.2.4 Príklad odstránenia anomálií

V bodoch 1) 2) 3) sú príklady na nesprávne (neefektívne) navrhnutú databázu a problémy, ktoré nastanú pri bežných operáciách na jej správu. Riešení, ktoré odstraňujú tieto problémy, je viac. Napríklad jednoduchou dekompozíciou do dvoch schém.

Firma_Data (Firma, Sidlo, Mesto) a Firma_Vyrobok (Firma, Vyrobok). Rozoberme si tento prípad. Pri mazaní výrobkov, ktoré firma vyrába, sa o nej zachovajú informácie. To je jedna z požadovaných vlastností. Ak chceme pridať firmu, ktorá

zatiaľ nič nevyrába, je to možné vytvorením záznamu v tabuľke Firma_Data. Zdá sa to byť všetko v poriadku. Ako však môžeme efektívne zmeniť názov firmy, poprípade názov výrobku? Museli by sme prejsť celú tabuľku Firma_Vyrobok a premenovať požadované informácie. To už až také optimálne nie je. Skúsme si preto navrhnúť iné schémy. Nech Firma_Data (IdFirmy, Firma, Sidlo, Mesto) Firma_Vyrobok (IDFirmy ,IDVyrobku) a Vyrobok (IdVyrobku, Vyrobok). Skontrolujme, či nám to nejako pomohlo. Zmena názvu firmy alebo výrobku sa dá jednoducho vykonať upravením príslušnej hodnoty v tabuľke Firma_Data alebo Vyrobok. Mazanie výrobkov je úprava tabuľky Firma_Vyrobok, pri ktorej sa nestrácajú žiadne informácie o firme ani o výrobku. Tento návrh teda vyhovuje všetkým požiadavkám na databázu.

Uvedenému postupu rozkladania schém sa hovorí normalizácia. Teda je to úprava schém do optimálneho tvaru.

Mohli sme si všimnúť, že hodnoty niektorých atribútov funkčne závisia na hodnotách iných atribútov. Napríklad, že ku každej firme existuje práve jedna adresa. Ide o istý vzťah medzi jednotlivými atribútmi, ktoré stoja za aktualizačnými anomáliami. V konečnom dôsledku závisí práve na programátorovi, aby navrhol jednotlivé závislosti podľa toho, aby odpovedali skutočnosti. Pojem funkčných závislostí zavediem a podrobnejšie vysvetlím v časti 2.1 Popis základných pojmov ako úvod do problematiky.

Otázkou však ostáva, ako navrhnúť schému, aby sme predišli problémom pri jej spravovaní.

Možné riešenia návrhu si predstavíme v nasledujúcej časti.

1.3 Navrhnutie schémy databázy

Modelovanie databázy môžeme robiť dvoma spôsobmi. Prvým je získať množinu schém prevodom z ER diagramu. Tu musíme znormalizovať každú tabuľku zvlášť na dodržanie normálnej formy. Nastáva riziko nadbytočného rozdrobenia databázy na príliš veľa tabuliek.

Druhým spôsobom, ktorý je použitý v programe, je chápanie modelovania celej databázy na úrovni globálnych atribútov a vytvorenie tzv. univerzálnej schémy databázy, teda jednej veľkej tabuľky, vrátane množiny globálne platných funkčných závislostí. Výhody takéhoto spôsobu sú, že normalizujeme iba raz pre celú databázu a je menšie riziko rozdrobenia na príliš veľa tabuliek. Modelovanie na úrovni atribútov je však menej intuitívne než ER modelovanie.

Taktiež skombinovaním týchto dvoch prístupov môžeme zhotoviť relačnú schému.

Teda najprv vytvoriť ER diagram, ten previesť do schém a postupne niektoré zlučovať, kým nedosiahneme požadovaný tvar.

Jediný spôsob normalizácie schém je rozdelenie schémy do viacerých tabuliek. Možnosti, ako toto rozdelenie vykonať, sú uvedené v časti nasledujúcej po oboznámení čitateľa so základnými pojmami. V nasledujúcom texte je názorný príklad, ktorý slúži na zadefinovanie pojmov, popis kontextu a pomáha lepšie pochopiť danú problematiku.

Kapitola 2.

Navrhovanie databázových schém

2.1 Popis základných pojmov ako úvod do problematiky

V každom návrhu chceme uchovávať informácie o nejakom objekte z reálneho sveta. Tento objekt (napr. osoba, firma, výrobok ...) budeme nazývať **entita.** Každá entita sa dá popísať množinou jej vlastností. Sú to informácie, hovoriace o jej druhu, stave, názve a pod., ktoré budeme chcieť uchovávať v databáze.

Každý stĺpec v tabuľke databázy obsahuje svoj názov (napr. Meno, Bydlisko, ICQ, RodneCislo...). Tieto názvy budeme nazývať **atribúty**. Sú to práve spomínané informácie o entitách. V rámci jednej databázy môže byť atribút obsiahnutý aj vo viacerých tabuľkách súčasne.

2.1.1 Funkčné závislosti

Funkčnou závislosťou budeme definovať vzťah medzi dvoma množinami atribútov.

Tieto závislosti vznikajú ešte pri návrhu databáze, takže nie je jednoznačne určené, ako presne budú vyzerať naše tabuľky. Dôležité je, že funkčné závislosti nemôžeme odvodzovať zo samotných dát danej databázy. Prístup je presne opačný. Funkčné závislosti platia všeobecne a databáza je nimi obmedzená. Ak by sme na to šli analýzou samotných dát, ľahko sa môže stať, že dostaneme nezmyselné funkčné závislosti. Napríklad majme schému Zamestnanec (Meno ,Rok_narodenia). Ak by

sme na základe konkrétnych dát zistili, že každý zamestnanec sa narodil v inom roku, mohli by sme sa domnievať, že platí Meno→ Rok_narodenia. Čo je, samozrejme, nezmysel.

Ukážeme si príklad na to, ako postupným odvodzovaním funkčných závislostí vieme zoptimalizovať návrh schémy a vyhnúť sa tým problémom pri jej správe.

Majme príklad, kde si ako atribúty zoberieme :

Meno, **P**riezvisko, **R**odne**C**islo, **B**ydlisko

Rozoberieme si teda vzťahy, ktoré medzi nimi platia. Jedine pomocou RodneCislo môžeme jednoznačne určiť, o ktorý riadok by sa jednalo v danej tabuľke. Takže naša závislosť by mohla vyzerať takto:

Rc → M, P, B

(pozn. **Rc → M, P, B** znamená, že k RC – RodneCislo je jednoznačne určené M - meno, taktiež P - priezvisko a B – bydlisko

iný zápis: **Rc → M; Rc → P; Rc → B;**)

Uvažujme situáciu v banke pri zriaďovaní účtu. Na pobočke banky s nejakým sídlom si klient založí účet nejakého druhu, na ktorý vloží peniaze.

Neefektívny návrh tabuľky by vyzeral takto:

CisloPobocky, Sidlo, RodneCislo, Bydlisko, Meno , Priezvisko, CisloUctu, DruhUctu, Stav

Je zrejmé, že málokto by navrhol takúto tabuľku ako štruktúru pre databázu. Tabuľka bude obsahovať vysokú redundanciu, čo by sťažovalo jej aktualizáciu, navyše, keď sme si už v predošlej úvahe odvodili **Rc → M, P, B.** Ako sa teda zmení schéma, ak prihliadame na túto funkčnú závislosť? Mohlo by to byť napríklad takto:

KLIENT (RodneCisloKlienta, Bydlisko, Meno , Priezvisko)

POBOCKA (CisloPobocky, Sidlo, RodneCisloKlienta, CisloUctu, DruhUctu, Stav)

(pozn. štruktúra znamená nasledovné: KLIENT– názov tabuľky; RodneCisloKlienta, Bydlisko, Meno , Priezvisko – atribúty obsiahnuté v tabuľke)

Skúsme sa teraz zamyslieť nad tabuľkou POBOCKA. Aké vzťahy tam platia? Určite to je CisloPobocky → Sidlo a CisloUctu → RodneCisloKlienta, DruhUctu, Stav.
Teda tabuľka bude v takomto stave obsahovať nadbytočné informácie o Sidle. Adresa pobočky je nemenná. Preto bude lepším riešením vytvoriť pre pobočku jednu tabuľku a pre informácie o užívateľskom účte druhú.

Celkový formálny popis databázy by mohol potom vyzerať takto:
POBOCKA (**C**islo**P**obocky, **S**idlo)
KLIENT (**R**odne**C**islo**K**lienta, **B**ydlisko, **M**eno , **P**riezvisko)
UCET (**C**islo**U**ctu, **C**islo**P**obocky, **R**odne**C**islo**K**lienta, **D**ruh**U**ctu, **S**tav)

Inými slovami, naša databáza by mohla obsahovať tri tabuľky s názvami POBOCKA, KLIENT, UCET.
Tabuľka POBOCKA má dva atribúty (stĺpce tabuľky) CisloPobocky a Sidlo (ďalej len Cp, Si).
Tabuľka KLIENT má taktiež dva atribúty RodneCisloKlienta, Bydlisko, Meno a Priezvisko (ďalej len Rck, B, M, P).
Tabuľka UCET má päť atribútov CisloUctu, CisloPobocky, RodneCisloKlienta, DruhUctu a Stav (Cu, Cp, Rck, Du, St).
Jednotlivé závislosti by potom boli Cp → Si, Rck →B,M,P a Cu → Cp, Rck, Du, St.

Nech množina **A** je množinou všetkých atribútov a množina **F** všetkých závislosti. Spolu tieto množiny dávajú **relačnú schému**.

2.1.2 Kľúč schémy

Ako príklad na vysvetlenie si vezmeme predošlý príklad a z neho atribút CisloPobocky. Jediné, čo z neho môžeme odvodiť, je Sidlo. Ale ak si vezmeme CisloUctu z tretej tabuľky, tak z neho môžeme odvodiť CisloPobocky,

RodneCisloKlienta, DruhUctu a Stav. Teda sme získali CisloPobocky, RodneCisloKlienta. Z prvých dvoch tabuliek môžeme pomocou nich odvodiť posledné atribúty: Sidlo, Bydlisko, Meno , Priezvisko.
To znamená, že pomocou atribútu (v našom prípade jedného) CisloUctu môžeme v danej databáze postupne odvodiť všetky atribúty. Tento atribút nazývame **kľúčový atribút**.

Formálna **definícia kľúča** schémy: Nech je daná relačná schéma R(A, F) a množina atribútov K taká, že $K \subseteq A$, potom K je kľúč schémy R, ak platia tieto vlastnosti:

a) $K \rightarrow A$
b) Neexistuje množina K', ktorá je vlastnou podmnožinou K, taká, že $K' \rightarrow A$.

Vysvetlenie:
Prvá podmienka hovorí o tom, že z kľúča je možné odvodiť všetky atribúty schémy. Druhá zabezpečuje minimalitu kľúča, teda odstránenie nadbytočných atribútov. Ak by platila iba podmienka a), ide o **nadkľúč**.

V predošlom príklade sme si zdôraznili význam funkčných závislostí. Sústreďme sa teraz viac na ich vlastnosti. Ich pozorovaním sa dajú všimnúť vzťahy, ktoré medzi nimi platia. Teda niektoré sa dajú odvodiť, iné zas platia vždy.

2.1.3 Armstrongove pravidlá

Zadefinujme si základné pojmy súvisiace s funkčnými závislosťami.
Majme R(A,F). Nech množiny atribútov X, Y, $Z \subseteq A$ a platí množina funkčných závislostí F

1) ak $Y \subseteq X$, potom $X \rightarrow Y$

Ide o triviálnu funkčnú závislosť. Napríklad $A,B \rightarrow A$ je triviálna.

2) ak $X \rightarrow Y$ a $Y \rightarrow Z$, potom $X \rightarrow Z$

Platí tranzitivita medzi závislosťami. Napríklad platí $A \rightarrow B,C$ a $B,C \rightarrow D$, teda platí aj $A \rightarrow D$.

3) ak $X \rightarrow Y$ a $X \rightarrow Z$, potom $X \rightarrow Y,Z$

Pravidlo kompozície závislostí.

4) ak $X \rightarrow Y,Z$, potom $X \rightarrow Y$ a $X \rightarrow Z$

Toto pravidlo sa nazýva dekompozícia.

Tieto pravidlá sa nazývajú **Armstrongove pravidlá** a majú tieto vlastnosti:

- sú korektné – čo z množiny F odvodíme, platí pre ľubovoľnú inštanciu R
- sú úplné – dajú sa nimi odvodiť všetky funkčné závislosti
  platné vo všetkých inštanciách R
- 1) ,2) ,3) sú nezávislé – odstránením akéhokoľvek z nich porušíme úplnosť
  (dekompozícia sa dá odvodiť z 1) a 2))

Elementárna funkčná závislosť je závislosť, ktorá má na pravej strane práve jeden atribút.

Na to, aby sme mohli začať navrhovať relačné schémy, je potrebné zadefinovať ešte niekoľko základných pojmov.

Majme $X, Y \subseteq A$. X a Y sú **funkčne ekvivalentné,** ak pomocou Armstrongových pravidiel vieme odvodiť funkčné závislosti $X \rightarrow Y$ a zároveň $Y \rightarrow X$. Teda platí $X \leftrightarrow Y$.

Funkčná ekvivalencia sa neskôr využije pri hľadaní funkčne ekvivalentných kľúčov v schémach, pri algoritme syntézy. Na príklade si ukážeme, ako funkčná ekvivalencia funguje.

Majme množinu atribútov X, Z, Y, P

a funkčné závislosti X, Z $\rightarrow$ Y; X, Y $\rightarrow$ P; P $\rightarrow$ Z

Použitím Armstrongových pravidiel dostávame:

Najprv podľa 1) platí X, Z $\rightarrow$ X a teda podľa 3) na základe platnosti X, Z $\rightarrow$Y platí X, Z $\rightarrow$Y, X

Druhým krokom je odvodenie podľa 2) na základe X,Y $\rightarrow$ P a P $\rightarrow$ Z pravidlo X,Y $\rightarrow$ Z a znovu použitím 1) platí X,Y $\rightarrow$ X. Teda dostávame X,Y $\rightarrow$ X, Z.

Overením definície funkčnej ekvivalencie pre množiny atribútov X, Z a X,Y, dostávame vzťah X, Z $\leftrightarrow$ X, Y a teda X, Z a X, Y sú **funkčne ekvivalentné.**

2.1.4 Funkčný a atribútový uzáver

Nesmieme však zabudnúť, že v danej schéme okrem závislostí explicitne uvedených, platia tiež závislosti logicky z nich vyplývajúce. Množina všetkých závislostí odvoditeľných podľa Armstrongových pravidiel z množiny funkčných závislostí sa nazýva **funkčný uzáver.** Označuje sa F^+.

Pokrytím množiny funkčných závislostí F nazveme množinu funkčných závislostí G takú, že $F^+ = G^+$. Ak je G množina elementárnych závislostí, ktorá vznikla z F pomocou dekompozície jej neelementárnych závislostí, hovoríme tomuto pokrytiu **kanonické**.

Dôležitým pojmom je **redundancia** funkčných závislostí. Môžeme ju definovať takto:

Funkčná závislosť *f* je redundantná v F, ak platí

$(F - \{f\})^+ = F^+$

To znamená, že ak by sme odstránili závislosť *f* z množiny závislostí F jej funkčný uzáver sa nezmení.

Alebo, inak povedané, po odstránení *f* z množiny funkčných závislostí je *f* odvoditeľná pomocou Armstrongových pravidiel z (F-*f*).

Neredundantné pokrytie je také pokrytie, ktoré neobsahuje redundantné závislosti. Je zjavné, že neredundantné pokrytie nie je určené jednoznačne. Záleží totiž na poradí v akom nadbytočné závislosti odoberáme. Na zisťovanie redundantných závislostí, poprípade atribútov v nich, nemusíme zakaždým vytvárať celý uzáver množiny závislostí. Pri takomto skúmaní nadbytočných závislostí nám pomôže výpočet atribútového uzáveru.

Atribútový uzáver množiny $X \subseteq A$ vzhľadom k množine funkčných závislostí F je množina všetkých atribútov funkčne závislých na X. Označujeme ju X^+.

Konkrétny algoritmus hľadania atribútového uzáveru si ukážeme neskôr, v časti popisujúcej spôsoby analýzy databázových schém (kap. 3). Skúsme si však načrtnúť dôležitosť a využitie atribútového uzáveru.

Predstavme si, že F obsahuje závislosť $X \rightarrow Y$ a v X sa nachádza taký atribút A, že platí $(X - A)^+ = X^+$. Vtedy môžeme povedať, že A je v X pre danú závislosť **redundantným atribútom**. Uvedeným spôsobom, s využitím algoritmu hľadania atribútového uzáveru, sa dajú odstrániť redundantné atribúty na ľavých stranách závislostí. Takéto závislosti sa nazývajú **redukované** a odpovedajúce pokrytie nazveme **minimálnym pokrytím**. Závislosti, ktoré nie sú redukované, sa nazývajú **čiastočné**.

Je dôležité si uvedomiť, že redukciou funkčných závislostí nestrácame žiadne znalosti o reálnom svete. Spätné odvodenie zredukovaných závislostí je možné použitím Armstrongových pravidiel. Tak napríklad uvažujme zredukovanie A, B $\rightarrow$C na závislosť B $\rightarrow$ C, teda odstránenie A z ľavej strany závislosti. Majme teda množinu atribútov A, B, C a zredukovanú závislosť B $\rightarrow$ C. Pridaním triviálnej funkčnej závislosti A, B $\rightarrow$ B a použitím tranzitivity dostávame pôvodnú závislosť A, B $\rightarrow$ C.

Dôležité upozornenie pri odstraňovaní redundantných informácií je, že ho nemôžeme vykonávať v ľubovoľnom poradí. Pre získanie minimálneho pokrytia je

nutné najskôr odstrániť redundantné atribúty z jednotlivých závislostí a až potom redundantné závislosti.

2.1.5 Algoritmus príslušnosti

Často potrebujeme zistiť, či je niektorá elementárna závislosť obsiahnutá v uzávere niektorej množiny závislostí. Teda vyriešiť problém $\{X \rightarrow Y\} \in F^+$.

F^+ môžeme vytvoriť tak, že vezmeme niektorú množinu pravidiel a odvodzujeme závislosti tak dlho, kým sa výsledná množina zväčšuje. Veľkosť F^+ je však všeobecne exponenciálna vzhľadom k počtu atribútov v schéme a jeho generovanie je neprehľadné a zložité.

Preto je výhodné, keď nemusíme fyzicky F^+ generovať, ale môžeme aplikovať tzv. **algoritmus príslušnosti** (membership algorithm), ktorý určí, či závislosť $X \rightarrow C$ je prvkom F^+. Priebeh algoritmu sa dá popísať tak, že generujeme atribútový uzáver X^+ vzhľadom k F a skúmame, či X^+ obsahuje C.

Algoritmus príslušnosti

1. $X^+ := X$
2. Opakovane (pokiaľ sa X^+ zväčšuje) prechádzame všetky pravidlá z F a vždy, keď ľavá strana niektorého pravidla je podmnožinou dosiaľ vytvoreného X^+, pridáme pravú stranu tohto pravidla do X^+
3. Ak je C prvkom výsledného X^+, je odpoveď na otázku príslušnosti kladná, inak záporná

Najdôležitejšia aplikácia algoritmu príslušnosti je pri eliminácii redundantných funkčných závislostí z F.

Pri generovaní schémy relačnej databázy je vhodné, aby (pri zachovaní F^+) bolo závislostí čo najmenej a boli čo "najmenšie", čiže čo najjednoduchšie z pohľadu počtu atribútov. Preto má odstraňovanie redundantných atribútov a závislostí veľký význam pri návrhu. Uveďme si algoritmus, ktorý **nájde neredundantné pokrytie**

pre množinu elementárnych funkčných závislostí, čiže algoritmus, ktorý odstraňuje nadbytočné závislosti. Je jednoduchý a využíva sa pri ňom algoritmus príslušnosti.

Vstup: F' nad množinou atribútov A a závislostí R(A)
Výstup: neredundantné pokrytie G

begin
G:=F'
for each $f \in G$ do
If $f \in (G-\{f\})^+$ then G:=G- {f}
end

Nájdenie **minimálneho pokrytia** pre množinu funkčných závislostí.

Vstup: F' nad množinou atribútov A a závislosti R(A)
Výstup: minimálne pokrytie G

begin
1. Zostroj pre F kanonické pokrytie F'
2. Uprav F' tak, aby všetky závislosti boli redukované
3. Odstráň redundantné funkčné závislosti (Algoritmus neredundantného pokrytia)

end

Uveďme si príklad na zostrojenie minimálneho pokrytia:
Majme schému R(A, B, C, D) a F = {A $\rightarrow$ A, C; B $\rightarrow$ A, B, C; D $\rightarrow$ A, B, C}. Skúsme pomocou algoritmu nájsť minimálne pokrytie. Postupujme podľa jednotlivých krokov

1. Kanonické pokrytie pozostáva z týchto elementárnych závislostí.

F' ={A $\rightarrow$ A; A $\rightarrow$ C ; B $\rightarrow$ A; B $\rightarrow$ B; B $\rightarrow$ C ; D $\rightarrow$A; D $\rightarrow$ B; D $\rightarrow$ C}

2. Všetky závislosti sú už redukované

3. Podľa algoritmu redukujeme redundantné závislosti

$A \rightarrow A$ je to triviálna závislosť a teda ju môžeme odstrániť

$A \rightarrow C$ ostáva pretože $A^+ = A$

$B \rightarrow A$ ostáva lebo $B^+ = B, C$

$B \rightarrow B$ je triviálna a teda redundantná

$B \rightarrow C$ môže byť eliminovaná pretože $B^+ = A, B, C$

$D \rightarrow A$ je redundantná $D^+ = D, A, B, C$

$D \rightarrow B$ ostáva pretože $D^+ = D, C$

$D \rightarrow C$ je odstránená lebo $A^+ = D, A, B, C$

Výsledné neredundantné, zároveň minimálne pokrytie množiny je $\{A \rightarrow C ; B \rightarrow A; D \rightarrow B\}$

2.2 Normálne formy

V uvedených príkladoch sme si mohli všimnúť, že sú to práve funkčné závislosti, ktoré spôsobujú za problémami, ale poskytujú aj možnosť ich riešenia, súvisiacich s návrhom databázovej schémy. Priradením jednej z normálnych foriem uvedených v tejto časti k schéme hovorí o jej vlastnostiach na základe funkčných závislostí, ktoré v schéme platia. Na konkrétnom príklade si ukážeme, ako môžu kľúč schémy a funkčné závislosti v nej ovplyvňovať aktualizačné anomálie.

Uvažujme schému popisujúcu program v kinách. Majme atribúty **N**azov**K**ina, **M**eno**F**ilmu, **A**dresa**K**ina, **D**atum.

Vieme, že kľúčom danej schémy je **N**azov**K**ina, **D**atum, a že každé kino má práve jednu adresu.

V takto navrhnutej schéme dochádza k značnej redundancii. Informácie o adresách kín sa totiž zbytočne opakujú vo viacerých n-ticiach. Ďalšou nevýhodou je možná strata informácií. Ak sa v kine nič nepremieta, strácame informácie o jeho adrese a názve. V prípade, že sa vytvorilo nové kino, v ktorom sa ešte nepremieta,

nemôžme ho do databázy uložiť. Nevieme totiž dátum premietania, ktorý je kľúčovým atribútom a kľúč musí byť definovaný.
Takto navrhnutá schéma nám teda nevyhovuje. Navrhnime ju preto inak, a to s prihliadnutím na fakt, že každé kino má práve jednu adresu. Schéma bude vyzerať takto: **N**azov**K**ina, **M**eno**F**ilmu, **D**atum budú tvoriť jednu tabuľku a **N**azov**K**ina, **A**dresa**K**ina druhú. Pri takomto návrhu boli všetky popísané problémy odstránené. Dôvodom výskytu spomenutých anomálií v predošlej schéme, bola závislosť nekľúčového atribútu na podmnožine kľúča. Konkrétne teda závislosť **N**azov**K**ina → **A**dresa**K**ina. Nazvime to, že **A**dresa**K**ina bola na **kľúči závislá čiastočne**. No odstránením takýchto čiastočných závislostí nevyriešime všetky anomálie.

Uveďme teraz iný príklad. Majme schému Film, Herec, Narodnost, Rok.
Kľúčom je dvojica Film, Herec a tiež vieme, že každý herec má práve jednu národnosť.
Ako aj v predošlom prípade, dochádza tu k redundancii (Narodnost) a niektoré informácie sa nedajú reprezentovať (ak herec nehrá v žiadnom filme). Ak schému upravíme na (Herec, Narodnost) a (Film, Herec, Rok), odstránia sa spomenuté problémy. Dôvodom je tentoraz odstránenie tranzitívnej závislosti Film → Herec →Narodnost. Teda nekľúčový atribút bol tranzitívne závislý na kľúči. To viedlo k popísaným problémom. Pri návrhu preto musíme vhodne eliminovať tranzitivitu.

Musíme však ešte vylúčiť jeden prípad tranzitivity. Uvažujme schému RodneCislo, Herec, Narodnost. Táto schéma má dva kľúče Herec (vlastné meno herca) a RodneCislo. Teda platí RodneCislo → Herec a Herec → RodneCislo. Aj keď táto schéma obsahuje tranzitivitu, a to RodneCislo → Herec → Narodnost, nenastávajú tu žiadne anomálie. Nemalo by totiž zmysel uchovávať informácie o jednom objekte s dvoma kľúčmi v dvoch tabuľkách.

Nech X, Y sú podmnožiny A, C je jednotlivý atribút, ktorý sa nevyskytuje ani v X ani v Y. Nech ďalej platí X → Y → C a neplatí, že Y → X. Potom hovoríme, že C je **tranzitívne závislé** na X.

Teraz môžeme zadefinovať istý „protokol“, ktorý musí spĺňať schéma. Je to takzvaná tretia normálna forma.

2.2.1 Tretia normálna forma

Relačná schéma A je v **tretej normálnej forme** (ďalej 3NF), ak pre každú funkčnú závislosť Y $\rightarrow$ x, kde Y je podmnožina A a x je atribút z A, platí aspoň jedna z nasledujúcich podmienok:

1) závislosť je triviálna, tj. atribút x je obsiahnutý v Y
2) Y je nadkľúč schémy A
3) x je kľúčový atribút.

Inak povedané, schéma je v tretej normálnej forme, ak každý nekľúčový atribút schémy nie je tranzitívne závislý na žiadnom kľúči schémy. Ak je schéma v 3NF, neobsahuje ani žiadne čiastočné závislosti nekľúčových atribútov na kľúči. Tejto vlastnosti sa hovorí druhá normálna forma (ďalej 2NF). Ak by totiž pre kľúč K a jeho vlastnú podmnožinu K' platilo K'$\rightarrow$ C, kde C je nekľúčový atribút, potom tiež platí K $\rightarrow$ K' $\rightarrow$ C, čo je v rozpore z 3NF. Ak je teda schéma v 3NF, je zároveň aj v 2NF.

2.2.2 Boyce-Coddova normálna forma (BCFN)

Hovoríme, že schéma je v BCNF ak je každý atribút závislý na kľúči.
Presnejšie, v danej schéme R(A, F) platí pre každú závislosť X $\rightarrow$ *a* (kde X $\subseteq$ A, *a* $\in$ A) aspoň jedna z podmienok

- závislosť je triviálna
- X je nadkľúč

Podmienky sú rovnaké ako pri 3NF, ale bez poslednej možnosti (*a* je súčasť kľúča)

BCNF vychádza z funkčných zvislostí a vlastne rozširuje 3NF. Každá schéma, ktorá je v BCNF je tiež v 3NF, teda neobsahuje žiadne tranzitívne funkčné závislosti. Pri návrhu, tvorbe alebo optimalizácii relačnej schémy je dôležité dostať ju aspoň do 3NF.

2.3 Kritéria pre návrh relačnej schémy

V dosiaľ uvedených príkladoch sme pri návrhu vychádzali z jedného kritéria. Tým bolo odstránenie tzv. aktualizačných anomálií. Pojmy ako 3NF a BCNF riešia tento problém.

Vždy keď sme chceli dosiahnuť 3NF alebo BCNF, museli sme nahradiť jednu schému niekoľkými inými schémami. Toto nahradenie zachováva zatiaľ iba jednu vlastnosť: Ak je A množina atribútov pôvodnej schémy a A_i, pre i = 1, 2, 3, ..., n, n>1, je množina atribútov i-tej schémy, potom zjednotenie všetkých A_i je rovné A. Tento proces sa nazýva **dekompozícia schémy**. Po takomto rozdelení však môžu nastať iné problémy. Je to napríklad strata niektorých funkčných závislostí, alebo vytvorenie informácií, ktoré nie sú v pôvodnej tabuľke obsiahnuté. Takéto správanie je nežiaduce. Pre jeho odstránenie musí výsledná schéma spĺňať ďalšie predpoklady:

a) Výsledná schéma by mala mať rovnakú sémantiku ako pôvodná. Teda musia v nej byť pokryté všetky závislosti.
b) Výsledná schéma by mala obsahovať rovnaké informácie, ako by mala pôvodná tabuľka

Tieto predpoklady sa overujú až po použití niektorého z algoritmov, ktorý vykonáva dekompozíciu. V prípade ich nesplnenia sa výsledná schéma musí upraviť. Pre lepšie uvedomenie si dôležitosti týchto predpokladov sú v nasledujúcej časti podrobnejšie popísané a doplnené názornými príkladmi.

2.3.1 Pokrytie závislostí

Sémantika je v schéme vyjadrená pomocou funkčných závislostí. Cieľom je, aby pôvodná schéma a schémy získané jej dekompozíciou pokrývali rovnakú množinu funkčných závislostí.

Uvažujme príklad, kde atribúty sú Mesto, Ulica, PSC, a majme závislosti Mesto, Ulica $\rightarrow$ PSC a PSC $\rightarrow$ Mesto. Po prevedení dekompozície do 3NF dostávame schémy: Ulica, PSC a PSC, Mesto. Tu však nastáva problém. Síce závislosť PSC $\rightarrow$ Mesto je v schéme, ale Mesto, Ulica $\rightarrow$ PSC už nie. Ľavá strana závislosti sa nachádza v jednej schéme a pravá v druhej. Závislosť je teda nepokrytá.

Majme relačnú schému databáze R = {S(A,F)} a dekompozíciu R_1 = {R_i (A_i, F_i), 1 ≤ i ≤ n, n ≥ 1}. Povieme, že R_1 má **vlastnosť pokrytia závislosti**, ak X^+ je rovné zjednoteniu F_i.

Zatiaľ ale nie je jasné, ako vytvoríme, F_i. Sú to závislosti, ktoré platia v A_i, čiže také závislosti z F^+ (nie iba z F), ktoré sa dajú „premietnuť“ do A_i. Budeme ich nazývať **projekcie závislostí**. Projekcia F do A_i je definovaná ako množina funkčných závislostí

F_i = {X→Y | X→Y je v F^+ a X, Y sú podmnožiny A_i}

Je zrejmé, že spočítať projekciu podľa definície je náročné, pretože veľkosť F^+ môže byť exponenciálna vzhľadom na veľkosť F.

2.3.2 Bezstratové spojenie

Zaveďme si teraz formalizáciu druhého predpokladu reprezentácie. Problém rovnakých dát sa dá jednoducho popísať pomocou operácie zjednotenia. Nech R({X ∪ Y ∪ Z}, F), kde X, Y, Z sú množiny atribútov, je univerzálna schéma, kde Y → Z ∈ F. Potom dekompozícia R1({Y ∪ Z}, F1), R2({Y ∪ X}, F2) je bezstratová.

Príklad stratovej dekompozície:

Firma	Výrobok	Výrobokov / Deň
Ford	Ka	20 ks
Ford	Focus	15 ks
Peugeot	307	25 ks
Peugeot	207	25 ks

Po dekompozícii do tabuliek:

Firma	Výrobok
Ford	Ka
Ford	Focus
Peugeot	307
Peugeot	207

Firma	Výrobokov / Deň
Ford	20 ks
Ford	15 ks
Peugeot	25 ks

Rekonštrukciou prirodzeným spojením dostávame:

Firma	Výrobok	Výrobokov / Deň
Ford	Ka	20 ks
Ford	Ka	15 ks
Ford	Focus	15 ks
Ford	Focus	20 ks
Peugeot	207	25 ks
Peugeot	307	25 ks

Zvýraznené riadky obsahujú dáta, ktoré v pôvodnej tabuľke neboli.

Príklad bezstratovej dekompozície:

Firma	Sídlo	Počet zamestnancov
Mecom	Humenné	5 000
Bukóza	Vranov n. Topľou	3 000
US Steel	Košice	10 000
Tesla	Stropkov	1 000

Po dekompozícií do tabuliek:

Sídlo	Počet zamestnancov
Humenné	5 000
Vranov n. Topľou	3 000
Košice	10 000
Stropkov	1 000

Firma	Sídlo
Mecom	Humenné
Bukóza	Vranov n. Topľou
US Steel	Košice
Tesla	Stropkov

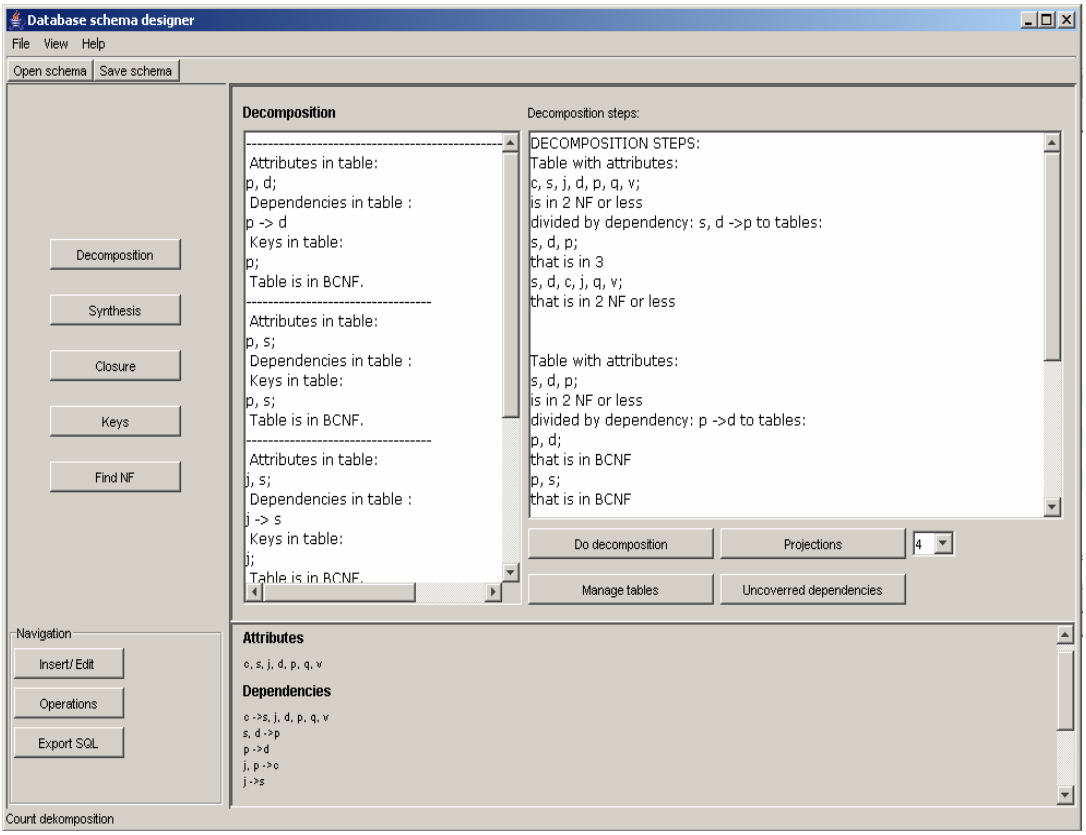
Ich prirodzeným spojením dostanem zhodné dáta z tými v pôvodnej tabuľke.

Kapitola 3

Aplikácia Database schema designer

3.1 O programe

Aplikácia je rozdelená na niekoľko častí. Je to menu a panel nástrojov pre rýchle spúšťanie, kde sa nachádzajú tlačidlá pre otvorenie a uloženie schém a výsledných tabuliek. V ľavej spodnej časti je navigačný panel. Pomocou jeho tlačidiel sa v ľavej hornej časti zobrazujú panely, v ktorých sú logicky rozdelené tlačidlá umožňujúce zadávanie, správu dát a volanie požadovaných operácií. V pravej spodnej časti aplikácie je zobrazený aktuálny zoznam zadaných atribútov a závislostí.



Program Database schema designer

Po spustení aplikácie má užívateľ niekoľko možností, ako s ňou pracovať. Buď zadá atribúty a závislostí medzi nimi, alebo ich načíta z už uloženého súboru. Po zadaní, poprípade po spracovaní vstupov, je aplikácia pripravená poskytnúť analýzu alebo navrhnutie rozvrhnutia tabuliek pre danú schému. Pre analýzu dát sú tu informácie o normálnej forme, kľúčoch, alebo možnosť nájsť atribútový uzáver, na základe vybranej množiny atribútov. Algoritmy dekompozície a syntézy, hľadajúce optimálne rozloženie schémy, sú taktiež dostupné. Po prevedení dekompozície je k dispozícii vyriešenie nepokrytých závislostí a prípadných projekcií. V syntéze sa po odstránení redundantných atribútov vytvoria výsledné tabuľky. Navrhnuté schémy jednou alebo druhou metódou, je možné ďalej spracovávať. Po spresnení informácií o obmedzeniach atribútov má užívateľ možnosť vygenerovať príkazy na vytvorenie tabuliek. V tabuľkách môžeme dodatočne upraviť primárny kľúč, doplniť cudzie kľúče, či celú schému uložiť do súboru.

Predtým, ako popíšeme vlastné algoritmy implementované v aplikácii, musíme si uvedomiť niektoré predpoklady. Jedným z nich je predpoklad o množine atribútov schémy, ktorú dekomponujeme. Tu zdôrazníme jednoznačnosť mien atribútov, čiže meno atribútu musí byť v celej schéme unikátne a pod jedným menom označujeme vždy ten istý atribút. Ďalším predpokladom je jednoznačnosť vzťahov, teda pre všetky množiny atribútov X existuje najviac jeden typ vzťahu. Napríklad VEDUCIPROJEKTU (Ucitel, Student) a UCI (Ucitel, Prednaska, Student) nevedú k splneniu tohto predpokladu, tj. zrejme nevznikla univerzálna schéma. Vedúci učitelia projektu a prednášajúci môžu byť k študentom v iných vzťahoch. Rovnaké funkčné závislosti však majú definovať v rôznych schémach tie isté funkcie.

3.2 Správa dát

3.2.1 Zadávanie atribútov

Po stlačení tlačidiel „Insert/Edit“ → „Insert“ → „Insert Attributes“ sa v hlavnom paneli zobrazí možnosť pridať do aplikácie nové atribúty. Na pravej strane sa taktiež zobrazí text s hlavnými zásadami a odporučeniami zameriavajúcimi

sa na tvar názvu atribútov. Po vyplnení požadovaných údajov sa potvrdením, stlačením tlačidla „Submit“, zobrazia novo zadané atribúty v dolnej časti aplikácie. Ak užívateľ naplní voľné políčka, použitím tlačidla „Insert more attributes>>“ sa uložia aktuálne zadané atribúty a uvoľní sa miesto pre zadanie nových.

3.2.2 Zadávanie závislostí

Možnosť zadávania závislostí sa zobrazí po stlačení „Insert/Edit“ → „Insert“ → „Insert Dependencies“. Ak má byť na niektorej zo strán závislosti viac atribútov, oddeľujú sa čiarkou. Podobne ako u zadávania atribútov sa zobrazia políčka, do ktorých môže užívateľ pridávať nové závislosti. V závislostiach sa musia na pravej a ľavej strane vyskytovať atribúty, ktoré sú už uložené. Ak sú po stlačení tlačidla „Submit“ pre potvrdenie zadaných závislostí v týchto závislostiach atribúty, ktoré nie sú ešte uložené, zobrazí sa okno s touto informáciou a možnosťou zmazať závislosť, alebo pridať nové atribúty. Taktiež sa tu nachádza tlačidlo „Insert more dependencies>>“ slúžiace na uloženie a uvoľnenie miesta pre zadávanie ďalších závislostí. Podobne ako u zadávania atribútov je na tomto paneli stručný popis, ako závislosti zadávať.

3.2.3 Editácia zadaných atribútov

Pre editáciu atribútov sa po stlačení „Insert/Edit“ → „Edit“ → „Edit Attributes“ zobrazí dialógové okno. V ňom je zoznam aktuálne uložených atribútov. Po výbere jedného z nich sa potvrdením zobrazí uložený atribút v editovateľnom políčku. Po vykonaných zmenách sa stlačením „Store changed attribute“ uloží zmena a obnoví sa zoznam atribútov. Ak sa upravovaný atribút nachádzal v jednej zo závislostí, zobrazí sa informácia, že program tento atribút nemá uložený, rovnaká ako u zadávania závislostí. Pre zmazanie niektorého z atribútov je potrebné vybrať atribút a stlačiť tlačidlo „Delete attribute“, atribút sa tiež zmaže pri zmazaní jeho mena pri editácii.

3.2.4 Editácia zadaných závislostí

Zadaním „Insert/Edit“ → „Edit“ → „Edit Dependencies“ sa podobne ako u editácii atribútov zobrazí dialógové okno. Použitie je skoro zhodné ako pri atribútoch. Po zadaní prázdnej pravej alebo ľavej strany závislosti a potvrdení uloženia sa závislosť vymaže.

3.2.5 Hromadné mazanie atribútov

Aplikácia poskytuje možnosť zmazať všetky atribúty naraz. Stlačením „Insert/Edit“ → „Delete“ → „Delete all attributes“ sa zobrazí dialógové okno, ktorého potvrdením, stlačením tlačidla „Ok“, sa odstránia všetky uložené atribúty. Zobrazí sa panel na vkladanie atribútov.

3.2.6 Hromadné mazanie závislostí

Funkcia, ktorá zmaže všetky závislosti stlačením „Insert/Edit“ → „Delete“ → „Delete all dependencies“, zobrazí podobný dialóg ako pri mazaní atribútov. Potvrdíme ho stlačením tlačítka „Ok“, a následne sa odstránia všetky uložené závislosti. Stlačenie „Cancel“ zruší dialógové okno a závislosti sa neodstránia. Zobrazí sa panel slúžiaci na vkladanie závislostí.

3.2.7 Vymazanie všetkých atribútov a závislostí

Stlačením „Insert/Edit“ → „Delete“ → „Delete all“ sa podobne ako u ostatných hromadných mazaní zobrazí okno s informáciou o mazaní. Jeho potvrdením sa odstránia všetky závislosti a atribúty. Zobrazí sa panel slúžiaci na vkladanie atribútov.

3.2.8 Ukladanie dát

Všetky zadané dáta v aplikácii sa dajú uložiť pre neskoršie znovupoužitie. V paneli nástrojov je tlačidlo „Save schema“ použitím ktorého sa otvorí dialógové okno s možnosťou zadania názvu súboru pre uloženie. Do súboru sa ukladajú nielen atribúty a závislosti, ale aj všetky tabuľky so svojimi nastaveniami. Súbory, do ktorých sa bude ukladať, môžu mať akúkoľvek príponu.

3.2.9 Načítavanie dát

Dáta uložené do súboru sa dajú načítať použitím tlačidla „Open schema“, ktoré je umiestnené v paneli nástrojov. Taktiež sa po kliknutí otvorí dialógové okno a v ňom si môže užívateľ vybrať súbor. Ak vyberie nesprávny súbor, aplikácia ho upozorní varovným dialógom. Všetky dáta, ktoré sú aktuálne v aplikácii budú stratené a nahradia sa dátami zo súboru.

3.3 Operácie

Užívateľovi aplikácia umožňuje zo zadanými atribútmi a závislosťami robiť rôzne operácie. V navigačnom paneli sa nachádza tlačidlo „Operations“, stlačením ktorého sa zobrazí panel s dostupnými operáciami.

Sú to prostriedky na analýzu danej schémy, konkrétne získanie informácií o jej normálnej forme, kľúčoch alebo možnosť zrátať na základe zvolenia množiny atribútov ich uzáver a tak zistiť či tvoria nadkľúč. Prejdime teraz k popisu implementovaných algoritmov.

3.3.1 Informácie o normálnej forme

Zobrazenie informácií o normálnej forme je dostupné v aplikácii kliknutím na „Operations“ → „Find NF“. Na dialógovom okne, ktoré sa ukáže, sa vyskytuje nielen záznam o normálnej forme, ale aj o atribútoch a závislostiach, ktoré tabuľka obsahuje. Keďže tabuľka je tvorená zo všetkých aktuálne uložených atribútov a závislostí, tento ich výpis môžme nájsť taktiež v spodnej časti aplikácie.

Vstup: Tabuľka T z uloženými atribútmi, závislosťami a kľúčmi
Výstup: Tabuľka T z uloženou hodnotou normálnej formy

V krokoch algoritmu prejdem všetky závislosti a postupne otestujem či vyhovujú pravidlám normálnych foriem a zapamätám si u závislosti maximálnu normálnu formu, ktorú neporušuje. Výsledná normálna forma tabuľky bude minimálna zapamätaná hodnota. Najnižšia možná normálna forma je 2NF, pretože algoritmy, ktoré sa v aplikácii používajú, rozkladajú schému aspoň do 3NF. Teda aplikácia nerozozná rozdiel medzi 2NF a 1NF formou a nastaví stále 2NF aj keď je to 1NF. Vo výpise to je zobrazené, že má 2NF alebo nižšiu normálnu formu.

3.3.2 Hľadanie kľúčov schémy

Hľadanie kľúčov schémy pozostávajúcej zo zadaných atribútov a závislostí je možné v aplikácii kliknutím na „Operations“ → „Keys“.

Na vyhľadanie kľúčov je použitý algoritmus, ktorý navrhli **Lucchesi a Osborn**. Algoritmus pozostáva z dvoch častí. Prvá je nájdenie akéhokoľvek kľúča schémy. Označme získaný kľúč K. V druhej časti potom prechádzame závislosti z F a pre každú závislosť X → Y takú, že prienik Y a K je neprázdny, vytvoríme množinu KX - Y. Ak táto množina nie je nadmnožinou už nájdeného kľúča, je kandidátom na ďalší kľúč – potrebujeme ho iba zredukovať na minimum. Pre každý novo nájdený kľúč postup opakujeme. Hľadanie skončí, keď v niektorom kroku žiaden nový kľúč nenájdeme.

Prvý kľúč nájdeme postupným redukovaním množiny všetkých atribútov na základe funkčných závislostí z F. Odstraňujú sa redundantné atribúty.

Algoritmus na hľadanie kľúčov schémy (prebrané z [5]):

Vstup: množina atribútov A,
množina závislostí F
Výstup: Množina všetkých kľúčov

algorithm ***GetAllKeys***(set of deps. F, set of atributes A) : **returns set of all keys**
Keys;
let all dependencies in F be non-trivial, i.e. replace every $X \rightarrow Y$ by $X \rightarrow (Y - X)$
K := *GetFirstKey*(F, A);
Keys := {K};
Done := **false**;
while Done = **false do**
Done := **true**;
for each $X \rightarrow Y \in F$ **do**
if $(Y \cap K \neq \varnothing$ **and** $\exists \neg K' \in$ Keys : K' $(K \cup X) - Y)$ **then**
K := *GetReducedAttributes*(F, $((K \cup X) - Y) \rightarrow A$);
Keys := Keys $\cup$ {K};
Done := **false**;
endfor
endwhile
return Keys;

3.3.3 Počítanie atribútového uzáveru

Atribútový uzáver zo zadanej množiny atribútov je možné zobraziť kliknutím na „Operations“ $\rightarrow$ „Closure“. Na paneli, ktorý sa zobrazí, sú zobrazené atribúty a pri nich zaškrtávacie políčka. Ak je počet zadaných atribútov väčší ako 10, zobrazia sa tiež tlačidlá, ktoré umožňujú zobrazenie nasledujúcich a predošlých atribútov. Teda užívateľ môže vyberať naraz z desiatich atribútov. Pri zobrazení ďalších atribútov sa uloží záznam o zaškrtnutí či nezaškrtnutí atribútu. Po vybraní

množiny atribútov, z ktorej sa bude uzáver rátať, kliknutím na „Find closure“ sa na pravej strane panela zobrazí informácia o výslednej množine uzáveru a tiež o tom, či zadaná množina tvorí nadkľúč v schéme, tj. či je uzáver totožný z celou množinou atribútov.

Nájdenie uzáveru zvolenej množiny atribútov (prebrané z [1])

Vstup: Množina X podmnožina A,

množina atribútov A,

množina závislostí F

Výstup: Množina atribútov tvoriaca uzáver x

V prípravnej časti algoritmu si vytvoríme ku každému atribútu zoznam závislostí, v ktorých sa nachádza na pravej strane. Taktiež si u každej závislosti zapamätáme počet atribútov, ktoré sú na pravej strane a ešte nie sú v množine tvoriacej uzáver. Vlastný algoritmus počítania uzáveru z takto pripravených dát prebieha nasledovne:

```
CL:=X;
For all A from CL do
    For each f from set of dependencies that contains A on right side
      Count(f):= Count(f)-1;
    If(Count(f)= =0) then
    Let f = Y→Z;
    CL:=CL∪ Z;
    fi;
od;
od;
```

Zložitosť je polynomiálna (O(m*n), kde **n** je počet atribútov a **m** počet vzťahov).

3.3.4 Syntéza

Úpravu schémy pomocou syntézy má užívateľ možnosť zhotoviť na paneli, ktorý sa zobrazí zadaním „Operations“ → „Synthesis“. Tento panel je rozdelený na štyri časti. V prvej sa zobrazuje výpis tabuliek, ktoré boli syntézou vytvorené. V druhej, ktorá je v spodnej časti, sú zobrazené jednotlivé kroky syntézy a teda i

výpis závislostí po odstránení redundancií na ľavých a potom na pravých stranách závislostí. V pravej časti panela sa zobrazia informácie o zmenách v tabuľkách po vyriešení problému projekcií a ekvivalentných kľúčov. Po prevedení algoritmu syntézy sa zobrazia tlačidlá, ktorými sa budú postupne riešiť problémy projekcie a ekvivalentných kľúčov. Ich názvy sú „Projection“ a „Equivalent keys“.
Výsledné tabuľky môžu byť ďalej spracovávané a editované v časti programu „Export SQL“ kliknutím na „Manage tables“. Touto akciou sa tabuľky uložia a zobrazí sa panel na ich editáciu a správu. Viď časť 3.3 Úprava tabuliek a generovanie vytvárajúcich skriptov.

Algoritmus syntézy

Algoritmus syntézy je spôsob, ako dosiahnuť aspoň 3NF.
Jednotlivé kroky syntézy sú nasledovné:

- nájdenie kľúčov a stanovenie normálnej formy
- odstránenie redundantných atribútov v závislostiach
- odstránenie redundantných závislostí
- spojenie závislostí s rovnakými ľavými stranami
- prvý návrh tabuliek
- vnorenie projekcie do „nadtabuliek“
- spojenie tabuliek s ekvivalentnými kľúčmi
- dodanie univerzálneho kľúča ako ďalšej schémy, ak ho žiadna schéma už neobsahuje

Obecne sa neuprednostňuje syntéza pred dekompozíciou, ani naopak. Každá ma svoje výhody aj nevýhody.

Príklad použitia syntézy (prebrané z [3]):
Majme množinu entít A = {c = Číslo zmluvy, s = Číslo dodávateľa, j = Číslo projektu, d = Číslo oddelenia, p = Číslo tovaru, q = Množstvo, v = Cena} a závislosti medzi nimi F = {c $\rightarrow$ s,j,d,p,q,v; s,d $\rightarrow$ p; p $\rightarrow$ d; j,p $\rightarrow$ c; j $\rightarrow$ s}
Redundantné atribúty na ľavej strane nie sú.
Po odstránení redundancií na pravých stranách
F = {c $\rightarrow$ j, d, q, v; s, d $\rightarrow$ p; p $\rightarrow$ d; j, p $\rightarrow$ c; j $\rightarrow$ s}

Boli odstránené c → s a c → p.

Teraz vytvorím výsledné tabuľky, podľa pravidla čo vzťah to tabuľka. Výsledná schéma tabuliek je (c, j, d, q, v), (s, d, p), (p, d), (j, p, c), (j, s) a každá z tabuliek je v BCNF.

Vyriešenie projekcie tabuliek:

Tabuľka (s, d, p) je nadtabuľkou (p, d). Preto ich môžme spojiť a vytvoriť novú tabuľku (s, d, p), kde platia závislosti s, d → p; p → d. Táto tabuľka však znižuje normálnu formu celej schémy, čiže na úkor zmenšenia rozdrobenia sa znížila normálna forma.

3.3.5 Dekompozícia

Ďalším už spomínaným spôsobom úpravy schémy je dekompozícia. K použitiu tohto algoritmu sa užívateľ môže dostať cez „Operations“ → „Decomposition“. Ľavá časť panela, ktorý sa zobrazí, slúži k výpisu výsledných tabuliek. V pravej časti panela sa zobrazujú informácie o jednotlivých krokoch dekompozície. Po prevedení dekompozície sa zobrazia tlačidlá, ktorými je možno vyriešiť prípadné problémy projekcie a nepokrytých závislostí.

Vlastný algoritmus dekompozície sa spustí kliknutím na „Execute decomposition“. Predtým však môže užívateľ zadať minimálnu formu, do ktorej chce schému rozkladať. To spraví výberom z ponuky vpravo dole na paneli, pričom má na výber 3, 4 alebo ----, čo znamená aspoň tretia, štvrtá alebo najvyššia možná normálna forma. Počas algoritmu sa zobrazujú informačné dialógové okná, v ktorých užívateľ volí závislosť, podľa ktorej sa má tabuľka ďalej rozkladať. V dialógovom okne sú kompletné informácie o tabuľke, ktorá je aktuálne spracovávaná, teda výpis atribútov a závislostí, ktoré neporušujú zadanú normálnu formu a samozrejme hlavne tých, ktoré ju porušujú. Pri nich je zobrazené číslo, ktoré značí normálnu formu, do ktorej sa tabuľka dostáva ponechaním závislosti v nej. V spodnej časti dialógu sú zobrazené tlačidlá „Ok“ a „Next“. Stlačením „Next“ sa zobrazí ďalšia možná závislosť, podľa ktorej sa dá tabuľka rozdeliť. Tlačidlo „Ok“ sa potvrdzuje výber závislosti a tabuľka sa rozloží. Ak je ďalšia tabuľka v nevyhovujúcej normálnej forme, proces výberu závislosti sa opakuje, pokiaľ nie sú všetky tabuľky v požadovanej normálnej forme.

Výsledné tabuľky môžu byť ďalej spracovávané a editované v časti programu „Export SQL“ tak, že sa klikne na „Manage tables“. Touto akciou sa uložené tabuľky nahradia novými a zobrazí sa panel na ich editáciu a správu. Viď časť 3.3 Úprava tabuliek a generovanie vytvárajúcich skriptov.

Cieľom návrhu relačnej databázovej schémy je splniť 3NF (alebo BCNF). Vo vytvorenej aplikácii je na to niekoľko možností. Jednou z nich je využiť implementovaný **algoritmus bezstratovej dekompozície**.

Bezstratová dekompozícia

Jedna z metód, ako dosiahnuť aspoň 3NF relačnej schémy je tzv. **dekompozícia**, t.j. rozklad danej relačnej schém na viacej schém, za účelom zvýšenia normálnej formy. Normálna forma databázovej schémy je určená najnižšou z normálnych foriem čiastkových relačných schém obsiahnutých v databázovej schéme.

Dekompozícia sa prevádza podľa nejakej funkčnej závislosti, ktorá nie je v 3NF. Z toho vyplývajú iste problémy pri prevedení dekompozície a to:

-nejednoznačný postup (záleží na výbere závislosti)
-niekedy nepokrytie závislostí
-niekedy viac tabuliek ako je nevyhnutné

Problémy nepokrytia všetkých funkčných závislostí a prípadných projekcií sa vyriešia po vykonaní samotného algoritmu. Zabezpečí sa tým odstránenie posledných dvoch uvedených problémov.

Model algoritmu použitého v programe:

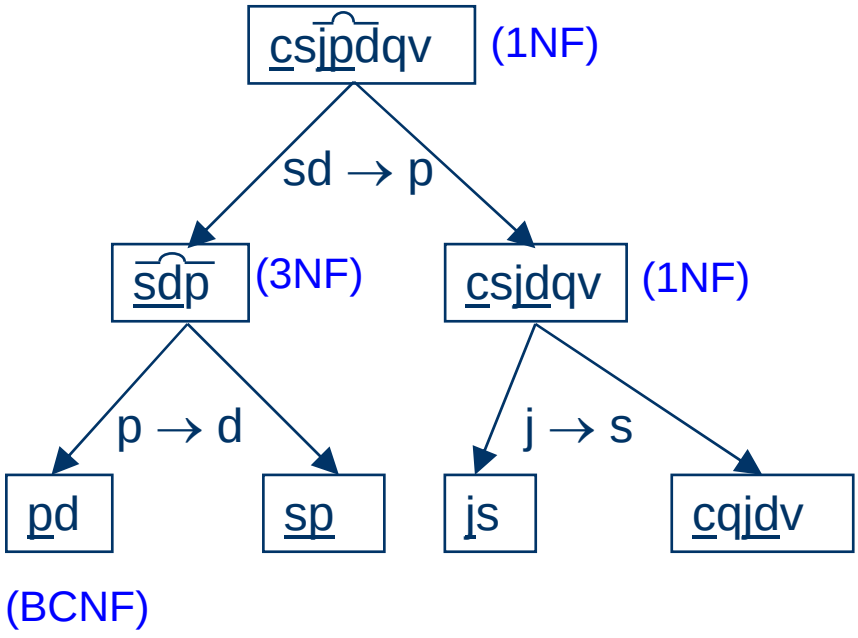
Dekompozícia do BCNF:
Vstup: R schéma (závislosti, vzťahy)
Kde F je odpovedajúca množina závislostí
(predp. že pre každú $X \rightarrow Y$ platí $X \cap Y = \varnothing$)
Výstup: {R1,…,Rk}schémy v BCNF
{F1,…,Fk}množiny závislostí

begin

```
result:={R};done:=false;vytvor(F+);
while not done do
begin
    if (result obsahuje Ri, které není v BCNF)
    then begin vyber netrivialnu závislost X → Y porušujucu BCNF;
        Rj:= {X,Y}; Ri:= (Ri \Y); result:= (result\Ri)∪Rj∪Ri;
        najdi (Fj, Fi)
    end
    else done:=true
end
end
```

Príklad použitia dekompozície:

Majme množinu atribútov A = {c = Číslo zmluvy, s = Číslo dodávateľa, j = Číslo projektu, d = Číslo oddelenia, p = Číslo tovaru, q = Množstvo, v = Cena} a závislostí medzi nimi F = {c → s, j, d, p, q, v; s ,d → p; p → d; j, p → c; j → s}
Algoritmom dekompozície postupne rozdeľujem tabuľku podľa závislostí porušujúcich normálnu formu.



Výsledná schéma tabuliek je v BCNF.

Po overení nepokrytých závislostí: Našli sa s, d→p ; c→p; Preto musím pridať nové tabuľky
(c, p) , ktorá je v BCNF a (s, d, p) v tretej normálnej forme.
Použitý príklad je popísaný v [4].

3.4 Úprava tabuliek a generovanie vytvárajúcich skriptov

Dodatočná úprava tabuliek, doplnenie dátových typov a obmedzení, ale aj zadávanie nových stĺpcov, či celých tabuliek a následné generovanie vytvárajúcich skriptov je funkcia, ktorú aplikácia poskytuje kliknutím na „Export SQL“.
Panel, ktorý sa zobrazí, obsahuje informácie o uložených tabuľkách. Pokiaľ bola vykonaná syntéza a následne stlačené „Manage tables“, budú zobrazené tabuľky, ktoré boli ňou vygenerované. To isté platí pre dekompozíciu a tlačidlo „Manage tables“ zobrazované na príslušnom paneli.
Pokiaľ je tabuliek viac ako 5, budú zobrazené tlačidlá na zobrazenie ďalších a predchádzajúcich piatich tabuliek. Taktiež sa zobrazujú tlačidlá na vymazanie všetkých tabuliek, pridanie novej tabuľky a vygenerovanie skriptov pre všetky tabuľky naraz.
Ako detail tabuľky je zobrazené jej meno a množina jej atribútov - stĺpcov. Meno je automaticky generované, aby bolo unikátne. Pri každej tabuľke je tiež tlačidlo „Manage table“. Jeho použitím sa zobrazí panel s detailom tabuľky, kde je možné meniť jej meno, pridávať stĺpce, meniť vlastnosti stĺpcov, či mazať ich.

Meno tabuľky sa zobrazí v hornej časti panela. Je editovateľné a potvrdením, stlačením tlačidla „Submit“, sa uloží aj s ostatnými zmenami. Stĺpce sa taktiež zobrazujú po piatich a použitím tlačidiel „More“ a „Previous“ sa postupne menia. Použitím „Add new table“ sa na koniec tabuľky pridá nový stĺpec s vygenerovaným unikátnym menom. Zmazanie stĺpca sa vykoná zmazaním jeho mena. Každý stĺpec má nastaviteľné hodnoty. Sú to meno, dátový typ, dĺžka dátového typu, obmedzenia na hodnotu null, či má byť hodnota unikátna, všeobecnú podmienku a implicitnú hodnotu pri vytvorení záznamu.
Dátový typ je možné vybrať z predvolených možností, jej dĺžku nastaviť zadaním hodnoty do vedľajšieho políčka, povolenie null hodnoty a unikátnosť zaškrtnutím

príslušnej hodnoty. Pre zadanie implicitnej hodnoty a všeobecnej podmienky aj s jej názvom sa po kliknutí na príslušné tlačidlo zobrazí dialógové okno. Ich zadané hodnoty sa nekontrolujú, takže je na užívateľovi, aby ich zadal správne.
Tlačidlom „Generate“ sa vygeneruje vytvárajúci SQL skript a zobrazí sa na výsledkovom paneli. Primárny kľúč sa určí na základe závislostí platných v tabuľke. Dodatočne sa dajú ešte meniť jeho hodnoty, ak má tabuľka viac ekvivalentných kľúčov, a to sa kliknutím na „Primary key“ v dialógovom okne, kde je zoznam prípustných primárnych kľúčov tabuľky.

Pridávanie cudzích kľúčov sa tak isto dá dodatočne riešiť kliknutím na „Foreign keys“. Po zobrazení dialógového okna je možné mazať existujúce a pridávať nové kľúče. Nové kľúče sa dajú pridať jednoduchým vybraním tabuľky, do ktorej sa chcem referovať, a zobrazia sa jej atribúty– stĺpce a primárny kľúč. Zadaním príslušných atribútov z práve upravovanej a odpovedajúceho počtu atribútov z referovanej tabuľky sa potvrdením cudzí kľúč uloží. Stlačením „Submit“ sa aplikujú zmeny na vytvárajúci skript a ten sa zobrazí upravený.

Kapitola 4

Záver

4.1 Výsledok práce

Táto práca ponúka čitateľovi zhrnuté informácie o problémoch návrhu databázových schém a program Database schema designer, ktorý má návrhárovi pomôcť pri ich zhotovení. Je v nej popísaný teoretický základ nevyhnutný pre zostrojenie návrhu databázy. Načrtnuté problémy názorne ilustrujú a dopĺňajú predstavu o reálnych situáciách s ktorými sa môže návrhár stretnúť. Popísané algoritmy poskytujú riešenia problémov pri návrhu a taktiež ponúkajú možnosť analýzy schém. Algoritmy implementované v programe zabezpečujú skonštruovanie výslednej schémy spĺňajúcej tretiu, po prípade Boyce-Coddova normálnu formu. To je zárukou odstránenia problémov s jej údržbou a zachovania požadovaných vlastností. Výstupom aplikácie sú taktiež SQL príkazy, použitím ktorých návrhár schému vytvorí. Toto rozšírenie programu je v praxi užitočné a prakticky dopĺňa využitie programu.

Zoznam literatúry

[1] Halaška, I., Pokorný, J., Valenta, M.: Databázové systémy: cvičení. Skripta, Vydavatelství ČVUT, 2002

[2] Pokorný, J.: Konstrukce databázových systémů. Skripta, Vydavatelství ČVUT, 2. vydání, 2004, 166 p.

[3] Materiály k prednáške A. Říhu Databázové systémy

[4] Materiály k prednáške T. Skopala Databázové systémy

[5] Materiály k prednáške Lenky Kebortovej Databázové systémy

[6] Pokorný, J. Databázová abeceda, Vydavatelství Science, Veletiny, 1998.